SmartBoard: A Unique System of Controlling Smart Devices Based around Writing

Vinay Ramesh

ABSTRACT

In the modern world, smart systems are often designed around the use of one's voice. This is easily seen with some of the bestselling systems on the market, like the Google Home smart hubs, or the Amazon Alexa smart devices. While this works well for a good majority of people, it faces the unique problem that it needs to take commands in through a voice. This inherently isolates consumers who want to buy smart devices, but cannot use their voice in ways which the smart devices require. In this paper, we propose a new style of smart home hubs, the Smart Board, a device that requires visual input for controlling smart home devices instead of vocal commands. It does so by using OCR to identify characters in the environment, and parsing these as commands for a smart home hub to execute. We created a sample device set up using these principles, and created mock functionality via controlling smart devices around the room, to great effect.

.

Author Keywords
Smart Devices, Internet of Things, Optical Character Recognition, Embedded Systems.

.

INTRODUCTION

Smart home hubs are arguably the fastest growing area of consumer Internet of Things (IoT) products. The reasoning behind such is rather simple, they provide a centralized platform for any consumer to control the rest of the smart devices in their home, and as such eliminates the need for multiple individual apps and mechanisms for controlling all parts of a smart home individually. However, these products, while powerful, generally come with a few simple problems that can be tough for people to overcome.

The first major problem is the requirement of voice. Almost all conventional smart home hubs require the use of one's voice as a deciding factor on what commands are being executed by the environment. While this works well for the most part, it can run into major issues in a few cases that aren't really accounted for by these smart home systems. The first is a rather simple one, in that there's only so far a voice can travel before it becomes inconvenient to project it further. This can become an issue because in order to have "coverage" across an entire house, one could often need multiple smart home hubs, and the cost of these can add up quickly, especially when combined with other smart devices around the home for them to integrate with, and the complexity of linking multiple smart home hubs together could be daunting for a casual IoT enthusiast.

Secondly, there's the problem that this completely alienates anybody who has problems with speech in the first place. This can be either anyone who has speech impediments or can't speak entirely. Because these smart home hubs require clear speech in order to pass commands off (and even then it's not necessarily always very accurate), people who have trouble speaking are naturally excluded from participating in IoT as a hobby.
In this paper, we attempt to create a smart home hub solution that answers both of these problems, by using the visual medium to transfer commands to the smart home hub, which are processed and the output is shown in the form of control of a smart light. The driving force behind this was to ensure that the smart home hub was accurate and relatively fast at recognizing commands. We tested this with multiple different handwriting styles and OCR methods to find out an ideal method of being able to constantly recognize handwriting within reason.

LITERATURE REFERENCE

As a first step in creating this system, I did research into other systems that might be similar, and tried to see if anything akin to this had already been done. For the most part, I could not find anything similar to a smart home hub that used OCR instead of voice. As such, I had to try and be creative with what literature I used. The first thing I decided to look into was handwriting recognition. On this front, I first looked into methods which people normally used for OCR, finding two papers which were really helpful. The first one, "A Recursive Otsu Thresholding Method for Scanned Document Binarization"[1] and "Strategies for handwritten words recognition using hidden Markov models"[2]. From these two papers, I realized the basic steps required to actually process handwritten text.

[1] Nina, O., Morse, B., & Barett, W. (n.d.). A Recursive Otsu Thresholding Method for Scanned Document Binarization. Retrieved 2020

[2] M. Gilloux, M. Leroux and J. -. Bertille, "Strategies for handwritten words recognition using hidden Markov models," Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR '93), Tsukuba Science City, Japan, 1993, pp. 299-304, doi: 10.1109/ICDAR.1993.395727.

After this, it was important to figure out backup options in case the Otsu Thresholding did not exactly work, and from that I looked into existing OCR implementations, specifically the google vision API and how it works. These were most of the literature that was used for the references of this paper, alongside additional forums on academic websites for the benefits and methods by which one could implement their own thresholding algorithms.

In particular, I looked into Otsu thresholding because that's the generally considered optimal way of thresholding images, but this will be covered in further detail in the section dedicated to the challenges that came with Optical Character Recognition.

SYSTEM SETUP

The first thing to consider for the system was what hardware was going to be used for this. Due to the limited resources at the time of undertaking this project (I was constrained by being limited to what I could find in my house, and what I could order off of Amazon realistically within a college student's budget). As such, the system was made with hardware limitations in mind. Given that, the actual hardware components used by this system are:

- ➔ Generic external webcam for laptop: This was purchased years ago, so the webcam quality was not the greatest (if this were to be repeated with a better budget, then a better webcam would have been used).
- ➔ Raspberry Pi 4: The remote computing component of this setup. A Raspberry Pi 4 was the device of choice due to its ability to run python, and familiarity with how the system functions. However, the methodologies described in this paper should apply to any microprocessor that can run python and has an input stream capable of taking input from a webcam.
- ➔ Yeelight Smart Lightbulb: The Yeelight smart lightbulb was used as the actuator for this system, where changes in the lightbulb would be easy to recognize (due to lighting changes in the room), and familiarity with the Yeelight API helped since it made streamlining the actuation process easier.
- ➔ Laptop: My laptop served as a "server" of sorts for the input captured by the raspberry pi. The actual server side setup was done in python, so it should be replicable on any system that can run python. My laptop was merely chosen for simplicity.

The setup of the system was designed in order to minimize the load on the raspberry pi itself, because from my own testing it was prone to overheating, even with heat sinks in place. As such, the system is a two-part setup, with the raspberry pi being the "eyes" of the system, and the laptop acting as the "brain". This will be covered a little bit more in the next section.

Architecture

As was outlined prior, the system was set up in two parts, with the raspberry pi-webcam component being the 'eyes' and the laptop being the 'brain'. In essence, the raspberry pi only contained the code required for preprocessing the image and turning it into legible text. From this, the text was sent to the computer via an If This Then That (IFTTT) connection. In particular, I used a simple REST API to push the text from the raspberry pi to the laptop over Wi-Fi. Once the laptop received the text, it used that to create changes with the Yeelight.

In addition, the OCR sometimes proved to be unreliable with the webcam camera, this was mostly due to angling being inconsistent (when worn on person) and the webcam camera not having the best resolution. As such, the raspberry pi also had a failsafe. If the system could not recognize text from the image using normal OCR methods, it would send the image to the laptop alongside a token. The laptop would then use an external API to process this image (In particular, the Google Vision API), and either cause a change if the result was valid, or do nothing if the result is an invalid command.

OPTICAL CHARACTER RECOGNITON

Because a large part of the system hinges on having the text in any given image be recognized correctly, the Optical Character Recognition component of this system is arguably the most important part. As such, this section will be attempting to explain the design choices behind which OCR methods were used, how they work in the actual system, and the results this shows.

Pytesseract and Thresholding

To start, the first OCR system used was Pytesseeract. This library is a wrapper for the Google Tesseract-OCR engine. The benefits of Pytesseract is that it is completely open source, and has arelatively quick result time. As such, it is a lightweight and very flexible option that can be sued for OCR, which is why it was the first choice for the system. However, this library comes with the downside that all of the image preprocessing has to be done manually beforehand since the OCR methods provided by the Pytesseract library do not work on raw images.

Image Preprocessing for OCR

To perform optical Character Recognition, as has been outlined in prior research such as "Strategies for handwritten word recognition…", there's often a few steps that are standard. To perform these, the OpenCV library was used as it is one of the most well documented python image processing libraries, and it's very versatile in its use cases.

From OpenCV, the raw images were first put through a greyscale filter. The reasoning for this is because the only significant aspects of an image, in this circumstance, is any text that may lie in it. As such we want to eliminate color and background noise so that the OCR algorithms applied by Pytesseract can recognize the text without much interference.

To further this, the greyscaled image is then passed through a gaussian blur filter. The reasoning for this is because passing an image through a gaussian blur smooths it out, and any individual dark spots get smoothed over and become 'less significant' for future steps of the process.

Finally, thresholding is applied to the image. What this does, in essence, is convert the image into a black and white feature vector, where pixels are either white (255, 255, 255) or black (0, 0, 0) depending on specific criterion that are passed into the thresholding algorithm at runtime. There's two main types of thresholding used for OCR in particular: binary thresholding and Otsu thresholding. Binary thresholding is rather simple, it involves using a simple condition to check the average value of any pixel, and if the pixel's average value falls below a certain threshold, it is marked as 0, otherwise it is marked as 255.

For the purposes of the SmartBoard system, binary thresholding is inefficient, however, because the dynamic lighting involved in moving a camera around household areas can produce images of text that do not work when a static threshold is applied to it. As such, Otsu thresholding is mandated.

Otsu thresholding and its pitfalls

Otsu's method is a thresholding algorithm that returns a single threshold that separates images into foreground and background (the black/white of the previous section).[3] The reason this works so much better than binary thresholding is because Otsu thresholding returns a unique threshold for any intensity of the image. As such, it's almost guaranteed to provide a result that separates the image into these two classes with some degree of reliability. Figure 1 shows a sample image at different steps of the image preprocessing process. As can be seen, the image is fairly consistent up until the first thresholding. At this stage, the image has been passed through a gaussian blur and has been greyscaled. After this it is passed through the Otsu algorithm and thresholded. The difference in clarity of the text between the pre-thresholding and thresholded image is clear to the human eye, however, this is also where we notice the first big problem with Otsu thresholding.

As can be seen in Figure 1, the thresholded image appears to have a 'shroud' around it. This is minimized a bit by applying adaptive thresholding (locally applying Otsu thresholding to different parts of the image), but the shroud still exists, and is problematic because it doesn't allow Pytesseract to effectively get the text from an image.

[3] Sezgin, M. (2004). Strategies for handwritten words recognition using hidden Markov models. Retrieved November, 2020

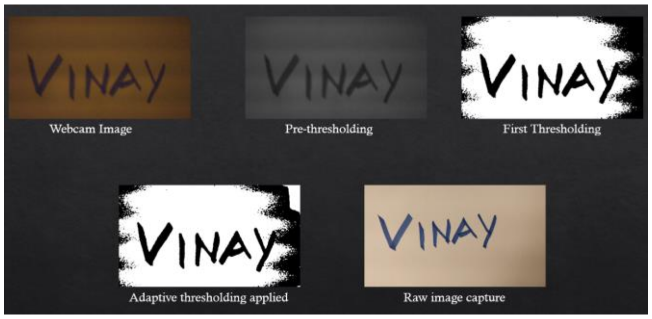

Figure 1: Sample webcam capture at different stages of preprocessing

Google Vision

To remedy the issue of shrouding caused by Otsu thresholding, a secondary OCR mechanism was implemented. The system uses the Google Vision API as a backup in case the initial Pytesseract reading does not return any significant output. In this case, the image (Webcam Image from Figure 1) is sent to the main server alongside a token that indicates that Pytesseract was unable to determine a result from the image. This image is then uploaded to the Google Vision hub, and a result is obtained from there.

For the most part, the Google Vision API was much more accurate than Pytesseract (this is covered in further detail in the Accuracy and Response Times section), but had a much slower response time due to having to wait for an external server to process the image. As a result, the Google Vision API was not favorable to use as the general use case for the SmartBoard. In addition, the Google Vision API is not a free service, and requires payment for usage. For the purposes of the SmartBoard prototype, this was covered by the free credits that come with using an academic email to set up a google cloud suite account, however this would not make for good scalability in an actual production environment, since the SmartBoard requires constant calls to an OCR method to identify what is being written in the environment.

ACCURACY AND RESPONSE TIMES

The metrics used for measuring the efficacy of the SmartBoard were the system's Accuracy, and the system's Response Time. Due to the nature of the system, these values had to be measured manually, specifically the response time, and the accuracy was measured based off of that. In particular, response times were measured by manually activating the program in front of text at various pre-measured distances, and then the time between the program activation and a change in the actuators was timed using a stopwatch, and recorded. These times were repeated for 3 different sample commands for the system ("LIGHT ON", "LIGHT RED" and "LIGHT OFF"). The commands were chosen because they had the most distinct visual change from each other, which made it easier to time the stopwatch. The raw data for this section is in Appendix 1 alongside the references.

Response Times

The response times for each of the commands at their respective distances are shown below. The distances in question were chosen through consultation with third parties (friends and family) about the distances they'd consider using a SmartBoard system at for day-to-day use, and adjusted based on stress testing it in the field. Below is the graph outlining the response times of the three commands tested.

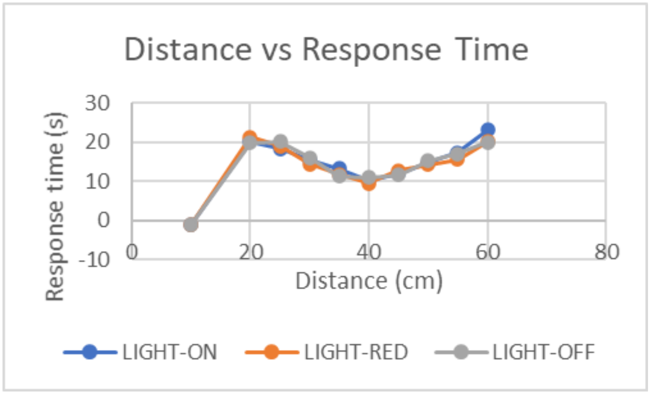

Figure 2: Distance vs Response time graph

In the graph, two distinct things can be noticed. The first is that when the camera is too close to the text (in this case below 10 centimeters), it produces an invalid output (-1), which indicates that the text cannot be scanned when it's too close. In addition, we can see that there appears to be a local minimum response time at 40cm distance between the text and the webcam used. With response times trending upwards after. The reasoning for this is likely due to camera focus and the clarity of the image captured. From some additional testing (using the webcam for its intended purpose), it seems like the focal distance of the webcam is a bit over 1 foot, which lines up with the response times recorded here. As such, the system performs at its best when the text to be scanned is located around where the focal distance of the camera being used to capture the text is.

Accuracy scores

Due to the nature of the SmartBoard system, measuring accuracy proved to be a rather complex task. Initially, it was assumed that a successful execution of a command could be counted as a part of an accuracy score, however the problem was soon encountered that given enough time, the system would output the correct command. As such, a different metric for measuring accuracy was devised:

The recorded response times were averaged to form a generic response time threshold for any given command at a given distance. From this, if a command was executed within the given response time, it was marked as a success. This was then repeated 100 times per command, and the number of successes per were recorded and plotted.

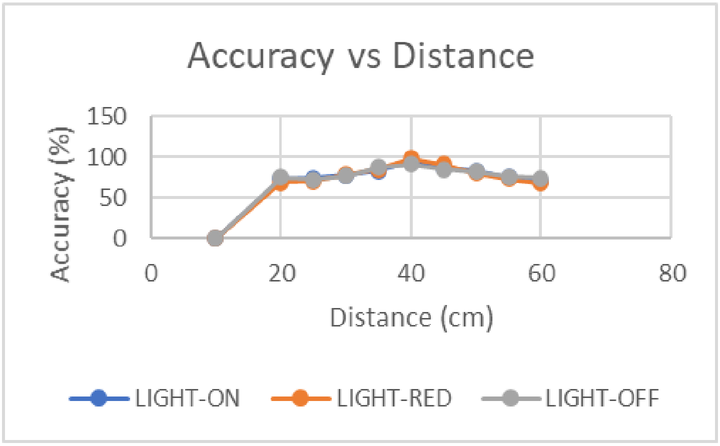

Figure 3: Accuracy vs Distance

The graph obtained is comparable to the response time graph that was shown earlier, with the local maxima being located at around 40cm, which is roughly the focal distance of the webcam. As such, this helps strengthen the conclusion that the system works at its best when the text is placed closer to the focal distance of the camera used for capturing the environment.

CONCLUSIONS

In this paper, the outlines for a system to create a smart home hub that utilizes writing instead of voice have been outlined. In essence, the methods behind OCR and a two-part architecture for the hub were outlined, alongside relevant response time and accuracy scores. The results of this prototyping showcased that it is indeed possible to have a functioning smart hub that utilizes writing instead of voice as its primary means of interacting with the user.

However, this system is not perfect, a lot of the limitations of the system were brought to light when actually designing and implementing it. From the response times, we can see that even the best response times average at 10 seconds, which is not ideal when dealing with writing because this could mean the user has to be stationary for that period of time, which can interrupt activities. When compared to traditional smart home hubs, where the voice integration does not require the user to interrupt their actions (unless they're speaking for some alternate reason), this can be inconvenient. However, there are fixes for this which can be explored in any potential future work, which are outlined below.

➔ The first fix is to have a better camera being used for the image capturing. Ideally one that auto-focuses before capturing the image. Since the system performs best when the image is at the focal distance of the camera, having one that auto-focuses means it's always going to be performing at its best.

- ➔ Possibly distributing the work between the server and the image capturing components better could lead to performance being better. In particular, sending a token and image to the system takes a while, so possibly having the google vision api code be on the raspberry pi as well could lead to performance improvements, however the downside is that the code on the image capturing component becomes less lightweight, and thus that part becomes bulkier.

Future Work

The system has a lot of potential to be expanded upon. For the purposes of the first prototype, only the simple task of controlling a smart light was considered. However, this has potential to be expanded upon to include far more things in a smart home ecosystem. In particular, integration with existing smart home systems (like an IFTTT applet or a google home system) could be plausible. In addition, it's likely possible to miniaturize the image capturing component of the system further, allowing it to become wearable and thus far more portable.

APPENDIX

Below is the data that was gathered as a part of the response time and accuracy score analysis:

Table 1: Distance vs Response Time

Distances	RT LON	RT LRED	RT LOFF
10	-1	-1	-1
20	20.12	21.33	19.83
25	18.43	19.01	20.21
30	15.42	14.37	16.04
35	13.23	11.87	11.25
40	10.33	9.54	10.99
45	11.94	9.54	11.64
50	14.73	14.22	15.28
55	17.32	15.6	17.04
60	23.11	20.1	19.99

For the above table,

RT LON = response time for command LIGHT ON,

RT LRED = response time for command LIGHT RED,

and RT LOFF = response time for command LIGHT OFF.

Table 2: Distance vs Accuracy Score

Distances	A LON	A LRED	A LOFF
10	0	0	0
20	73	69	75
25	74	71	72
30	78	79	77
35	83	85	88
40	94	98	91
45	87	91	85
50	83	80	82
55	75	73	76
60	71	68	74

Similar to the response time table:
A LON: accuracy for command LIGHT ON,

A RED: accuracy for command LIGHT RED, and

A LOFF: accuracy for command LIGHT OFF.

REFERENCES

1. Nina, O., Morse, B., & Barett, W. (n.d.). A Recursive Otsu Thresholding Method for Scanned Document Binarization. Retrieved 2020
2. M. Gilloux, M. Leroux and J. -. Bertille, "Strategies for handwritten words recognition using hidden Markov models," Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR '93), Tsukuba Science City, Japan, 1993, pp. 299-304, doi: 10.1109/ICDAR.1993.395727.
3. Sezgin, M. (2004). Strategies for handwritten words recognition using hidden Markov models. Retrieved November, 2020